

A common scheme for running NLO ep event generators

Thomas Hadig^a, Gavin McCance^b

^a I. Physikalisches. Institut, RWTH Aachen, D-52056 Aachen, Germany

^b ZEUS Collaboration, mccance@zow.desy.de

Abstract: In this article we present a generic interface to several next-to-leading order cross-section programs. This enables the user to implement his/her code once and make cross-checks with different programs.

1 Introduction

Next-to-leading order QCD Monte Carlo programs are used for comparisons of QCD perturbation theory and experimental data. Nowadays four programs are available, which allow user defined observables; these are MEPJET[1], DISENT[2], DISASTER++[3], and JETVIP¹[4]. With the availability of different programs cross-checks become important. In order to make a comparison, the user code has to be implemented for each program. This procedure is susceptible to bugs and updating of several versions needs a decent revision control system.

To eliminate this additional source of error, a common scheme was developed and is presented here. The scheme consists of three independent parts, which are described in the following sections.

Section 2 introduces the steering card. This file is read at the beginning of each calculation and sets up the most important parameters. The user has the ability to add additional parameters to steer his own user routines.

The interface to the user routines is described in section 3. This code calculates the observables the user is interested in. For most standard procedures, e.g. performing boosts to different frames or calculating the number of jets, special functions are available in a library. These functions are explained in detail in section 5.

A more detailed version of this manual including the full description of the function library and an example are available with the source code at www.desy.de/~heramc/proceedings [6].

¹JETVIP is currently not available in the common NLO library.

2 Steering Card

At the beginning of the calculation a steering card is read from standard input to set up the input parameters. This allows the easy modification of the most important starting values even after compilation.

While the generator's adjustable parameters are known, this is not true for the user code. Therefore an easy interface is provided, which allows the user to expand the set of steering card parameters.

2.1 Basic Structure of the Steering Card

The basic structure of the steering card is equivalent to that of most Monte Carlo generators. The steering card consists of several banks, labeled by a character string of four characters. The number of banks in a steering card is not limited.

Each bank consists of an unique four character label, an integer version number and an arbitrary number of entries (including none).

An entry consists of an identifier and the value of the field. The value can be of type integer, real or string, where string stands for a character constant of arbitrary length.

Comments can be marked by an asterisk (*) in the first column or by an exclamation mark (!) anywhere in a line. If a comment mark is found the rest of the line is ignored.

2.2 Predefined Banks

Many parameters, like the particle density function and the number of events, have to be set for all programs, these parameters are collected and can be set by the use of a steering card bank called **MOCA**.

Below you see the complete **MOCA** bank. Some of the entries are described in detail below.

```
MOCA 0           ! NLO Monte Carlo steering card, Version 0
  'TYPE' 2        ! NLO program (1=Mepjet, 2=Disent, 3=Disaster++)
  'NEVE' 10000     ! Number of Events (required)
  'Q2MI' 10.       ! (D=100.) Q**2 min
  'Q2MA' 100.      ! (D=4000.) Q**2 max
  'XMIN' 0.        ! (D=0.) x min
  'XMAX' 1.        ! (D=1.) x max
  'XIMI' 0.        ! (D=0.) xi min
  'XIMA' 1.        ! (D=1.) xi max
  'YMIN' 0.        ! (D=0.) y min
```

```

'YMAX' 0.7          ! (D=1.) y max
'W2MI' 0.           ! (D=0.) Minimum W**2
'W2MA' 100000.      ! (D=100000.) Maximum W**2
* PDFL and PDFH give the PDF libs used for LO (L,SET.eq.0) and
* NLO (H,SET.ne.0) calculations, should be identical for MRS and
* of corresponding type for CTEQ and GRV,
* example : PDFL=5005 (GRV94 LO) and PDFH=5006 (GRV94 NLO) for
* MSbar scheme and PDFH=5007 (GRV94 NLO) for DIS scheme
'PDFL' 3035         ! (D=3035=MRSH MSbar) PDF lib,
'PDFH' 3035         !   NGROUP*1000+NSET as in pdflib manual
'ECMS' 90200.       ! (D=90200.=27.5*820.*4.) CMS energy
'EP ' 820.          ! (D=820) Proton energy
'LEPT' 1            ! (D=1) incoming lepton (1: e-, 2: e+)
* The following values (upto * end) are given in the lab frame
'ELMI' 11.          ! (D=0.) Minimum energy of scattered electron
* 'ELMA' 11.         ! (D=1000.) Maximum energy of scattered e-
'TLMI' 150.         ! (D=0.) Minimum angle of scattered electron
'TLMA' 172.5        ! (D=180.) Maximum angle of scattered e-
* end
'NFL ' 5            ! (D=5) number of flavours
'SFQ2' 1.           ! (D=1.) factorization scale factor for Q**2
'SRQ2' 1.           ! (D=1.) renormalization scale factor for Q**2
'SFKT' 0.           ! (D=0.) factorization scale factor for kt**2
'SRKT' 0.           ! (D=0.) renormalization scale factor for kt**2
'SFPT' 0.           ! (D=0.) factorization scale factor for pt**2
'SRPT' 0.           ! (D=0.) renormalization scale factor for pt**2
'SFCO' 0.           ! (D=0.) factorization scale factor for 1
'SRCO' 0.           ! (D=0.) renormalization scale factor for 1
'IAEL' 0            ! (D=0) switch for alpha_electromagnetic
                    ! 0 : fixed, 1 : running
'AELM' 0.00729735308 ! (D=0.00729735308) alpha_electromagnetic
'MASS' 0.15         ! (D=0.15) mass of strange quark in GeV
'MASC' 1.4          ! (D=1.4) mass of charm quark in GeV
'MASB' 4.4          ! (D=4.4) mass of bottom quark in GeV
'MAST' 174          ! (D=174) mass of top quark in GeV
'ALOO' 2            ! (D=2) number of loops for alpha_s running,
                    ! -1: pdf routine, 0: fixed, 1|2|3: 1|2|3 loop
'LAM4' 0.3          ! (D=0.3) Lambda_4_MSbar for running alphas
                    ! or alphas for fixed alphas

```

- ITYPE is coded as follows :

ITYPE=0 : unknown

ITYPE=1 : MEPJET

ITYPE=2 : DISENT

ITYPE=3 : DISASTER++

- NEVE is the only field required for each run. It gives the number of events to be generated².
- PDFL and PDFH specify the parton density functions used. Two values are given in order to enable the distinction of leading order and next-to-leading order processes. The format of each of the values is NGROUP * 1000 + NSET where NGROUP and NSET are given by the PDFLIB.
- The renormalization and factorization scale can be set using the formula

$$\mu_s^2 = f_{Q2}Q^2 + f_{PT}p_t^2 + f_{KT}k_t^2 + f_{CO}$$

where the factors f_{xy} are set with the steering parameter Ssxy.

- IAEL and AELM steer the running of $\alpha_{\text{el.mag.}}$ and ALOO and LAM4 steer the running of α_s accordingly. When ALOO is set to -1, the α_s calculation included in the pdf library is used. If it is 0, $\alpha_s \equiv \text{LAM4}$ for every scale, otherwise LAM4 corresponds to $\Lambda_{\overline{MS}}^4$ and the masses MASx are used for calculating $\Lambda_{\overline{MS}}^{(3,5,6)}$.

Some parameters are still program specific, therefore for each generator an additional steering card exists. Below you can find the steering card of DISASTER++[3] and DISENT[2].

```
DISE 0          ! DISENT specific steering card
* 'SEDL' -1     ! (12345) Lower seed value for random generator
                ! if set to negative value a time-depended
                !   value will be used
* 'SEDH' -1     ! (678900) Higher seed value for random generator
                ! if set to negative value a process-depended
                !   value will be used
'SCHE' 0        ! (D=0) 0 : MSbar, 1 : DIS
                ! Please note : the PDF has to be chosen equivalent
'NP01' 2        ! (D=2) The parameters NP01 and NP02 are internal
'NP02' 4        ! (D=4) parameters used by Disent, please refer
                ! to the program manual
DISA 0          ! DISASTER++ specific steering card
```

²It should be noted for DISASTER++, however, that the product FFIN * POINTS determines the number of events generated. Using the library default values, DISASTER++ will produce 50% more events than DISENT.

```

'BORN' 2          ! (D=2) number of particles in born term
'PROC' 0          ! (D=1) order of process (0 : L0, 1 : NLO)
'FPRE' 1.5        ! (D=1.5) factor for # points in preparation run
'FFIN' 1.5        ! (D=1.5) factor for # points in final run

MEPJ 0            ! MEPJET specific steering card
'BOSO' 1          ! exchange boson (1:gamma, 2:Z, 3:gammaZ, 4:W)
'BORN' 2          ! (D=2) number of particles in born term
'PROC' 0          ! (D=1) order of process (0 : L0, 1 : NLO)
'NPRO' 100        ! (D=100) process number (quark+gluon, unpolarized)
                  ! see Mepjet manual for details (iproc)
'IMAS' 0          ! (D=0) massive calculation (0=no, 1=yes)
'IPDF' 4          ! (D=4) pdf crossing function
'ITER' 4          ! (D=4) number of iterations for vegas adaptation
'SMIN' 0.1        ! (D=0.1) smin cone size
'PTIN' 0.1        ! (D=0.1) minimal p_t in lab frame for partons
'AMIN' -10.       ! (D=-10) minimal pseudo rapidity in lab for partons
'AMAX' 10.        ! (D= 10) maximal pseudo rapidity in lab for partons
'FILE' 'mepjet'   ! base name for vegas grid files

```

Please note, that due to the usage of crossing functions the parton density functions for Mepjet is not set with PDFL, PDFH, but by the IPDF switch using the crossing functions found in the pdfcross directory.

2.3 User Functions for Steering Card Access

In order to add an additional parameter to the steering card, two steps are required. First the parameter has to be initialized before the steering card is read using one of the following routines :

- call SCARINT(bank,identifier,value) for integer parameters
- call SCARREAL(bank,identifier,value) for double precision parameters
- call SCARCHAR(bank,identifier,value) for string parameters

where **bank** is the four letter bank name, **identifier** is the one to four character identifier and the **value** is the default value of the type corresponding to the type of the parameter.

After the steering card has been read, the user can retrieve the current parameter value. To do so, three functions are available :

- value = GCARINT(bank,identifier) for integer parameters

- `value = GCARREAL(bank,identifier)` for double precision parameters
- `value = GCARCHAR(bank,identifier)` for string parameters

where `bank` and `identifier` correspond to the values given above and `value` will be the one given in the steering card or the default parameter, if the corresponding bank or identifier in the bank was not set in the steering card. Values and banks may appear more than once in a steering card, but only the last value is stored and can be retrieved.

In principle the get routines could be called each time one of the values is needed, but since this requires several string comparisons, it is rather slow. The recommended procedure is to get the values once and store them in a FORTRAN common block.

3 User Routines

In this section, the routines that have to be supplied by the user in order to run the programs are described. Therefore the general scheme is sketched here. The number of user routines is quite large, but most routines can be replaced by empty or short routines if no special action is needed.

Figure 1 shows the calling tree of the relevant functions and subroutines. Routines printed in `typewriter` have to be supplied by the user and are described in detail below, functions in *italic* and roman are provided by the corresponding cross-section Monte Carlo and the library, respectively.

3.1 Initialization and Termination Routines

Several initialization and termination routines are called to allow the user to predefine values, to open files, or to book histograms. None of these subroutines take an argument. In the following each routine is described.

`disinit` is called only once at the beginning of the program. This is the place to initialize the users steering card values.

After the steering card has been read, `init` is called. Here the parameter values can be copied to a common block, as proposed at the end of section 2. Also output files should be opened here.

Corresponding to these initialization routines a termination routine `disterm` exists. This function is called once at the end of the run. Here all files should be closed.

In addition to these called-once routines an additional set of init and term routines exists. The `evtinit` subroutine is called before a new event is generated and `evtterm` afterwards. Since all contributions to the observable of one event have to be added before the error can be calculated[5], the main summation can only be done in those routines.

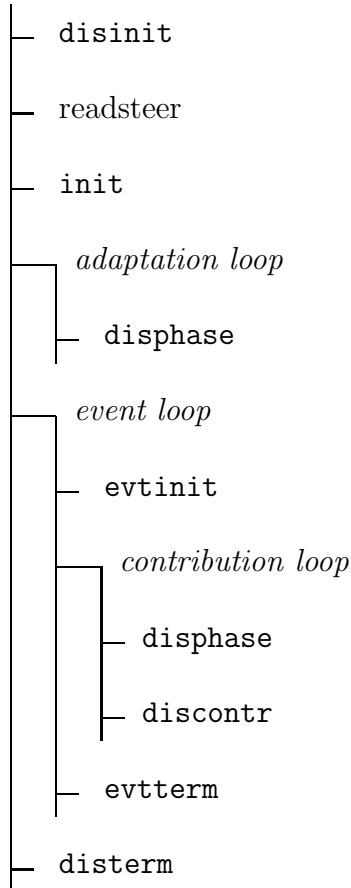


Figure 1: *Overview of the calling tree.*

Please note that the `evtterm` routine will not be called if an event has to be dropped due to technical reasons (e.g. after a floating point exception). See the web version of this manual [6] for an example.

3.2 Main User Routines

For each event a number of phase spaces is generated, each of these phase spaces can have several contributions. The main idea is to calculate your observables once for each phase space `iphasno` and add all weights of the corresponding contributions to the event weight. The calculation of the observables is done in `disphase` and the summing is done in `discontr`. The user has to take care that the observables are saved in a common block and are available when `discontr` is called. It is sufficient to allocate space for up to 30 different phase spaces.

In addition to the actual event sampling, some programs perform an adaptation loop to adapt the phase space to the region that is actually used. In this additional loop `disphase`

is called only. The adaptation is then done according to the return value.

The syntax is

```
function disphase (iphasno, npar, ps, iadapt)

double precision disphase
integer          iphasno, npar, iadapt
double precision ps(4,10)
```

and

```
subroutine discontr (iphasno, ntype, nalp, nalps, n2pi,
+   ilognf, jmin, jmax, weight )

integer          iphasno, ntype, nalp, nalps, n2pi
integer          ilognf, jmin, jmax
double precision weight(-13:11)
```

where the arguments have the following meaning :

- **iphasno** gives the number of the phase space.
- **npar** is the number of outgoing particles (excluding the scattered lepton)
- **ntype** is the type of the contribution

ntype = -1 unknown

ntype = 0 tree level

ntype = 1 subtraction counterterm

ntype = 2 finite virtual term

ntype = 3 finite collinear term

ntype = 4 renormalization term

- **nalp** and **nalps** are the orders of α_{elm} and α_s , respectively. **n2pi** gives the power of an additional factor of $\frac{1}{2\pi}$.
- **ilognf** selects one of the following factors :

ilognf=0 : $\lambda = 1$

ilognf=1 : $\lambda = \log \left(\frac{\mu_r^2}{Q^2} \right)$

$$\text{ilognf}=2 : \lambda = N_f \log \left(\frac{\mu_r^2}{Q^2} \right)$$

$$\text{ilognf}=3 : \lambda = \log \left(\frac{\mu_f^2}{Q^2} \right)$$

$$\text{ilognf}=4 : \lambda = N_f \log \left(\frac{\mu_f^2}{Q^2} \right)$$

- **ps**(i, j) defines the phase space. The array contains several particles distinguished by the second index j .

$j = 1$: incoming proton

$j = 2$: incoming lepton

$j = 3$: boson

$j = 4$: outgoing lepton

$j = 5$: outgoing proton remnant

$j = 6$: incoming parton

$j = 7, \dots, 10$: outgoing partons

The first index gives the 4-vector components, where $i = 1$ corresponds to the energy, and $i = 2, 3, 4$ to the x, y , and z components of the momentum, respectively.

- **iadapt** is set to 1 during the adaptation loop. The value returned by **disphase** is used to adapt the integration and sampling phase space. If a negative is returned, a default adaptation calculation is used.
- **weight** is an array containing several weights. For each contribution only a specific range in this array, defined by **jmin** and **jmax** is valid. Each weight has to be multiplied by a factor ρ_i and then all weights have to be summed. The factor ρ_i for the weight i depends on the particle density functions f_α for the different flavours α and is calculated by :

$$i = -8 \dots -13 : f_{\bar{\alpha}} \text{ with } \alpha = u(-8), d, s, c, b, t(-13).$$

$$i = -7 : f_g$$

$$i = -1 \dots -6 : f_\alpha \text{ with } \alpha = u(-6), d, s, c, b, t(-1).$$

$$i = 0 : 1$$

$$i = 1, 4 : \sum_{\alpha=1}^{N_f} Q_\alpha^2 f_\alpha$$

$$i = 2, 5 : \sum_{\alpha=1}^{N_f} Q_\alpha^2 f_{\bar{\alpha}}$$

$$\begin{aligned}
i = 3 & : \sum_{\alpha=1}^{N_f} Q_{\alpha}^2 f_g \\
i = 6 & : (1 - N_f) \sum_{\alpha=1}^{N_f} Q_{\alpha}^2 f_{\alpha} \\
i = 7 & : (1 - N_f) \sum_{\alpha=1}^{N_f} Q_{\alpha}^2 f_{\bar{\alpha}} \\
i = 8 & : \sum_{\alpha=1}^{N_f} f_{\alpha} \sum_{\beta=1, \beta \neq \alpha}^{N_f} Q_{\beta}^2 \\
i = 9 & : \sum_{\alpha=1}^{N_f} f_{\bar{\alpha}} \sum_{\beta=1, \beta \neq \alpha}^{N_f} Q_{\beta}^2 \\
i = 10 & : \sum_{\alpha=1}^{N_f} f_{\alpha} Q_{\alpha} \sum_{\beta=1, \beta \neq \alpha}^{N_f} Q_{\beta} \\
i = 11 & : \sum_{\alpha=1}^{N_f} f_{\alpha} Q_{\bar{\alpha}} \sum_{\beta=1, \beta \neq \alpha}^{N_f} Q_{\beta}
\end{aligned}$$

The full weight of one contribution is then

$$w_{\text{contribution}} = \lambda \alpha_s^{\text{ialps}} \alpha_{\text{elm}}^{\text{ialp}} \left(\frac{1}{2\pi} \right)^{n2\pi} \sum_{i=j\text{min}}^{j\text{max}} \text{weight}(i) \rho_i$$

This somewhat complicated procedure can be performed by a library function `double wsum (nord, nalp, nalps, n2pi, ilognf, jmin, jmax, weight, fscale, rscale, alphas)`. See section 2 of the web version[6] for an example. `nord` selects the leading or higher order particle density function (see steering card values `PDFL` and `PDFH`). `fscale` and `rscale` are the input values for the factorization and renormalization scales, respectively. `alphas` is an output parameter returning the value of α_s at this phase space point.

Additional information for experienced users : For DISASTER++ all phase spaces for one event are generated at the beginning of the event and the `disphase` routine is called for each phase space, before the first call to `discontr` is made. The `ntype` information is 0 for tree level and -1 otherwise. In DISENT the sequence is different, for each contribution `disphase` and `discontr` are called right after each other. The `ntype` information is fully available. The adaptation loop is performed in DISASTER++, only.

3.3 Common Blocks

Some common blocks are provided to the `disphase` and `discontr` routines. They contain the steering card values for the predefined bank and the event kinematics.

Here are the definitions³ :

```

INTEGER          MOCIVERS, NEV, ITYPE, IALELM, NPDFL, NPDFH
INTEGER          NFL, ALOOP, LEPTON
DOUBLE PRECISION Q2MIN, Q2MAX, XMIN, XMAX, YMIN, YMAX, S
DOUBLE PRECISION ELMIN, ELMAX, TLMIN, TLMAX, W2MIN, W2MAX
DOUBLE PRECISION SRQ2, SFQ2, SRPT, SFPT, SRKT, SFKT, SRCO
DOUBLE PRECISION SFCO, DEFAEM, XIMIN, XIMAX, EP
DOUBLE PRECISION LAMBDA3, LAMBDA4, LAMBDA5, LAMBDA6
DOUBLE PRECISION MASSS, MASSC, MASSB, MASST, LEPTON

```

```

COMMON /STERMOCA/MOCIVERS, NEV, ITYPE, IALELM, DEFAEM,
+ Q2MIN, Q2MAX, XMIN, XMAX, YMIN, YMAX, S, EP, NPDFL,
+ NPDFH, ELMIN, ELMAX, TLMIN, TLMAX, W2MIN, W2MAX,
+ SRQ2, SFQ2, SRPT, SFPT, SRKT, SFKT, SRCO, SFCO,
+ XIMIN, XIMAX, NFL, ALOOP,
+ LAMBDA3, LAMBDA4, LAMBDA5, LAMBDA6,
+ MASSS, MASSC, MASSB, MASST

```

and

```

INTEGER          DISEVERS, ISEEDL, ISEEDH, NSCHEME
INTEGER          NP01, NP02

```

```

COMMON /STERDISE/DISEVERS, ISEEDL, ISEEDH, NSCHEME,
+ NP01, NP02

```

```

INTEGER          DISAVERS, IPROC, IBORN
DOUBLE PRECISION FPRE, FFIN

```

```

COMMON /STERDISA/DISAVERS, IPROC, FPRE, FFIN, IBORN

```

```

INTEGER MEPJVERS, IMBOSO, IMPROC, IMMASS, IMPDF,
+ IMBORN, IMEPROC, IMITER
DOUBLE PRECISION RMSMIN, PTMIN_DEF, YMIN_DEF, YMAX_DEF
COMMON /STERMEPJ/MEPJVERS, IMBOSO,
+ IMBORN, IMEPROC, IMITER, IMPDF,
+ RMSMIN, IMPROC, IMMASS,
+ PTMIN_DEF, YMIN_DEF, YMAX_DEF

```

³For a detailed discussion see section 2

where the values correspond to the steering card parameters.

The additional KINE common block defines the event kinematics with some invariants and some energies in the laboratory frame of reference.

```
DOUBLE PRECISION Q2, XB, YB, W2, SUMKT2, SUMPT2,
+               ESCELE, THSCELE, EE, XI, ALPHA

COMMON /KINE/EE, XI, Q2, XB, YB, W2, ESCELE, THSCELE,
+   ALPHA, SUMKT2, SUMPT2
```

4 Interface to HzTool

HzTool[7] is a package that provides code to compare data plots published by the HERA experiments to Monte Carlo programs. Some of the plots could also be compared to next-to-leading order calculations and an interface from this library to HzTool is presented here.

Note that several observables are not applicable in next-to-leading order programs, such as particle or track multiplicities, and others would require to add hadronisation effects to the calculation. Therefore the comparison is a priori limited to a few observables.

HzTool reads data from the HEPEVT common block, a standard format for high energy physics Monte Carlo programs. In addition, HzTool expects that every call contains the full information for one event with a corresponding weight. Since different contributions to one event have to use a special error treatment, a decent error calculation without changing all HzTool routines is not possible.

The restrictions implied are therefore:

- Only observables available to next-to-leading programs allow comparisons.
- Be especially aware of cuts that spoil cancelations of divergencies[8].
- Errors calculated by HzTool are not valid.
- Differences according to non-perturbative effects can be expected.
- Some NLO programs might give unusable results, e.g. dijet rates need the simultaneous calculation of $\mathcal{O}(1)$ and $\mathcal{O}(\alpha_s)$ tree level diagrams, which is e.g. not possible in Disaster++.

To use the HzTool interface, add the file `nlolib/src/hztool/hzhep.f` to your source. This routine provides implementations to the standard user routines and therefore replaces the interface specified in section 3 and figure 1 by the one given in figure 2.

The routines that have to be provided by the user are `hzinit()`, `hzterm()`, and `hzuser()`. These routines should call the `hzxxxxx` routines with `iflag` equal to 1, 3, and 2 respectively. See the HzTool manual [7] for more information.

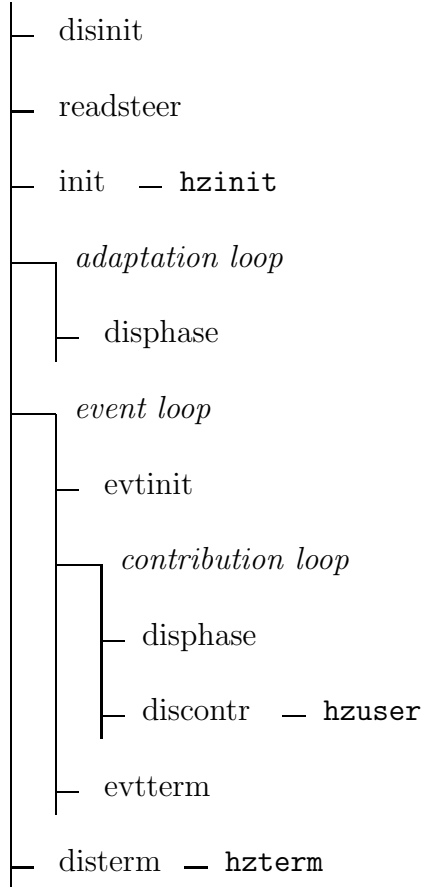


Figure 2: Overview of the calling tree when using *HzTool*.

4.1 Note for authors of `hzxxxxx` routines

In order to work with this library, `hzxxxxx` routines have to fulfil some additional requirements.

- Add the common block `HZNLO` to your routine.
- Initialize the variable `NLO` to 0 in the `iflag.eq.1` section.
- Add an `if` clause to every `hbook` fill or weight summation line in order to check, whether the current event counts for the desired process, e.g. check for total cross sections, whether the process is a $\mathcal{O}(1)$ born term or a $\mathcal{O}(\alpha_s)$ correction. This check can easily be done using the variables defined in the `HZNLO` common block.

The `HZNLO` common block is defined as follows:

```

      INTEGER NLO, TOT, DIJET, TRIPJET, QUADJET
      COMMON /HZNLO/NLO, TOT, DIJET, TRIPJET, QUADJET

```

An example might look like:

```

      IF (iflag.eq.1) THEN
C... init step
      NLO=0
      ELSE IF (iflag.eq.2) THEN
C... fill step

C... make some phase space cuts

CTH add total x-section
      IF ((NLO.eq.1).AND.(TOT.ne.1)) GOTO 11112
      nall=nall+xw
11112  CONTINUE

C... run some jet algorithm and make some cuts on dijet events

      IF (twojet) THEN
      IF ((NLO.eq.1).AND.(DIJET.ne.1)) GOTO 11114
      nall2=nall2+xw
      call hfill(121,x,0.,xw)
      call hfill(131,q2,0.,xw)
11114  CONTINUE
      ENDIF
      ENDIF

```

5 Function Library

In addition to the unified interface a set of subroutines is defined. Please, see the full list below.

- jet algorithms
 - JADE
 - k_t
 - longitudinal boost-invariant k_t
- event shape variables

- Lorentz boosts
- support routines for calculation of momentum, angles, masses, etc.

For the full documentation of most routines see the web version[6].

5.1 Jet Algorithms

Several jet algorithms exist. The calling sequence for those is

```
double precision P(4,*), YCUT, SCALE, VEC(4,*)
integer          NPA, IRECOM, NUM

call <algo>jet(P,NPA,YCUT,SCALE,I,IRECOM,NUM,VEC)
```

where <algo> is replaced by the name of the algorithm (i.e. one of `jade`, `kt`).

The common arguments are :

- **P** the input particles,
- **NPA** the number of input particles,
- **VEC** the output jet axes and energies,
- **NUM** the number of jets and
- **IRECOM** gives the recombination scheme for two particles (see the description of the support routine `vadd` in section 5.4 for more details).

The remaining arguments are algorithm dependent. Please refer to the individual manual. The basic idea is, that **SCALE** corresponds to a scaling factor and **YCUT** corresponds to a cut value. This is for example the scale for the invariant mass and the maximum mass over scale value for the `jade` algorithm and the minimum jet E_t and the maximal cone radius for the cone algorithm. **I** is an additional input value with no predefined meaning.

For the `KTCLUS` package, a slightly different calling sequence is used :

```
SUBROUTINE KTINCJET(P,NPA,PTMIN,NUM,V)

DOUBLE PRECISION P(4,*),V(4,*),PTMIN
INTEGER NUM,NPA

SUBROUTINE KTCLUSJET(P,NPA,SCALE,YCUT,NUM,V)

DOUBLE PRECISION P(4,*),V(4,*),SCALE,YCUT
INTEGER NUM,NPA
```

where the arguments correspond to the arguments explained in the beginning of the section.

5.2 Lorentz Boost

Two routines are provided to perform Lorentz boosts to other reference frames.

```

SUBROUTINE boost1 (V,BR,BZ,BXZ,IDIR,IERR)

DOUBLE PRECISION V(4),BR(4),BZ(4),BXZ(4)
INTEGER          IDIR,IERR

SUBROUTINE boost (P,V,N,BR,BZ,BXZ,IDIR,IERR)

DOUBLE PRECISION P(4,*),V(4,*),BR(4),BZ(4),BXZ(4)
INTEGER          N,IDIR,IERR

```

boost1 boosts one vector, given in array **V** into the new frame, where **V** will be used for input and output. **boost** can boost **N** vectors given in array **P** into the new frame. Here the original vectors are conserved and the new vectors are filled in array **V**. **boost** should be preferred, when more than one vector is to be boosted, since the calculation of the rotation matrices is done only once for all vectors.

The boost is specified by three 4-vectors, which will have the following characteristics in the boosted frame : **BR** will be the 0-vector, **BZ** will point in z-direction and **BXZ** will lie in the x-z-plane. If **IDIR** is zero a normal boost will be performed, if **IDIR** is one, the boost is reverted such that boosting the returned particles in **V** with the given boost vectors (and **IDIR=0**) would result in the original particles **P**.

If an error occurs during boosting **IERR** will contain a non-zero value.

The routines

```

SUBROUTINE blab2br1 (V,IERR)
SUBROUTINE blab2br (P,O,N,IERR)
SUBROUTINE bbr2lab1 (V,IERR)
SUBROUTINE bbr2lab (P,O,N,IERR)

DOUBLE PRECISION P(4,*),O(4,*),V(4)
INTEGER          N,IERR

```

perform the boosting of one or several vectors from the H1 laboratory frame to the Breit frame and vice versa.

5.3 Event Shape Routines

A single routine is provided to calculate a variety of event shape variables.

```
SUBROUTINE getevtshape (NPAR,PS,SCALE,SHAPE,SUME)

DOUBLE PRECISION PS(4,10),SCALE,SHAPE(NOVAR),SUME
INTEGER          NPAR, NOVAR
PARAMETER        (NOVAR=20)    ! the current number of variables
```

where the input parameters are

- **NPAR** the number of input particles,
- **PS(4,11)** the four vectors **PS()** from **disphase**,
- **SCALE** the scale to normalise to for some variables, usually Q .

the subroutine outputs

- **SUME** the total energy in the current region of the Breit frame,
- **SHAPE(NOVAR)** and array with all the event shapes. They are indexed by a letter code eg. **SHAPE(itz)** for current jet thrust.

The table below gives the indexes for all 14 of the available event shapes. The remaining 6 variables in **SHAPE(20)** are internal or obsolete and should not be used.

The index variables are available by including the file **index.inc**. The routine is designed to be called from within **disphase** and passed the common library four-vectors **ps()** directly. No cut on the total energy in the current region (to ensure infrared safety) is performed within the routine; this quantity is returned and any cut is the responsibility of the user routine.

SHAPE(itz)	Current jet thrust
SHAPE(itz2)	2nd moment of current jet thrust
SHAPE(itm)	Thrust axis thrust
SHAPE(itm2)	2nd moment of thrust axis thrust
SHAPE(ijb)	Current jet broadening
SHAPE(ijb2)	2nd moment of current jet broadening
SHAPE(ijm)	Current jet mass
SHAPE(ic4)	C-parameter (normalised to energy)
SHAPE(ic5)	C-parameter (normalised to momentum)
SHAPE(ic6)	C-parameter (normalised to $scale/2$)
SHAPE(ied)	Energy deficit
SHAPE(iob)	Oblateness
SHAPE(itr)	Thrust to resultant axis
SHAPE(ijbr)	Jet broadening to resultant axis

5.4 Support Routines

A collection of small functions are available. Most of the routines come in two versions, one that takes a vector and one that takes an array of vectors and an index as arguments.

- **vcopy** (A,I,B,J) copies the i'th vector of array A to the j'th vector of array B.
- **vadd**(A,I,B,J,IREFCOM) adds the j'th vector in array B to the i'th vector in array A. The available recombination schemes are :

IREFCOM=1 E/JADE : $\mathbf{p} := \mathbf{p}_a + \mathbf{p}_b$.

IREFCOM=2 E0 : $E := E_a + E_b, \vec{p} := \frac{|\vec{p}|}{E}(\vec{p}_a + \vec{p}_b)$ (momentum rescaling)

IREFCOM=3 P : $\vec{p} := (\vec{p}_a + \vec{p}_b), E = |\vec{p}|$ (energy rescaling)

IREFCOM=4 Snowmass : $p_t = p_{t,a} + p_{t,b}, \eta = \frac{1}{p_t}(\eta_a p_{t,a} + \eta_b p_{t,b}), \varphi = \frac{1}{p_t}(\varphi_a p_{t,a} + \varphi_b p_{t,b}), E = |\vec{p}|$

- **P = ATH** (A,I) calculates the theta angle (in radian) of the i'th vector of array A wrt. the +z-axis.
- **P = VP2** (V) and **P = AP2**(A,I) returns the momentum squared of the particle V and A(i), respectively.
- **P = VMAS2** (V) and **P = AMAS2**(A,I) returns the invariant mass squared of the particle V and A(i), respectively.

In the above functions and subroutines the variables are defined as follows :

```
DOUBLE PRECISION A(4,*), B(4,*), V(4), P
INTEGER          I,J,IREFCOM
```

6 Summary

In this paper we have presented a scheme for a complete library of next-to-leading order QCD programs. This library consists of three parts; a steering card mechanism, a unified interface for the user routines and an expandable set of FORTRAN library functions.

Acknowledgments

We would like to thank the authors for their support and appreciate the comments and suggestions on the program and this paper. We also thank the organizers and conveners of this workshop for the interesting topics.

References

- [1] E. Mirkes and D. Zeppenfeld; Phys. Lett. B380 (1996) 205, hep-ph/9511448.
- [2] S. Catani and M.H. Seymour; CERN-TH-96-029, Nucl. Phys. B485 (1997) 291-419, hep-ph/9605323.
- [3] D. Graudenz; Deeply inelastic hadronic final states: QCD corrections, 1997, *Preprint* PSI-PR-97-20 in: *Proceedings of the Ringberg workshop on new trends in HERA physics, May 1997*, hep-ph/9708362,
D. Graudenz; DISASTER++ *Version 1.0*, 1997, hep-ph/9710244,
D. Graudenz; DISASTER++ *Version 1.0.1 program manual* unpublished.
- [4] B. Pötter; JetViP 1.1: calculating one- and two-jet cross sections with virtual photons in NLO QCD, 1998, *Preprint* DESY 98-071, hep-ph/9806437,
- [5] Disent Program Manual, Version 0.0.
- [6] A more detailed version of the manual and the program source can be found at <http://www.desy.de/~heramc/proceedings>
- [7] T. Carli et.al., HzTool — A Package for Monte Carlo Generator - Data Comparisons at HERA, <http://dice2.desy.de/~h01rtc/hztool.html>
- [8] T. Hadig; Issues on NLO pQCD Programs, J. Phys. G: Nucl. Part. Phys. 25 (1999) 1470-1472, hep-ex/9907036.